CS 361 - Homework 01 - Program

Professor Mark W. Boady

1 Overview

There used to be a TV game show called **Let's Make a Deal** which ran for many years with host Monty Hall (1921-2017). You can learn more about it at `https://en.wikipedia.org/wiki/Let%27s_Make_a_Deal`. (Note: Professor Jeff Popyack came up with the original version of this homework.)

A typical situation in the game show was this: the stage is set up with three large doors labeled #1, #2, and #3. Behind two of the doors is a goat. Behind the other door is a new car.

You don't know which door has the car, but Monty Hall wants you to pick a door. Suppose you pick door #1. Monty Hall then opens one of the other doors (say, door #3) and shows you that there's a goat behind it. He then says: "I'll give you a chance to change your mind. Do you want to change your mind and pick door number 2?" After you make a decision, they open the door you finally picked and you find out whether you've won a new car or a new goat. To clarify: the locations of the car and the goats are fixed before your game starts. Monty does not get a chance to switch things around in the middle of the game. He just shows you that one of the choices you didn't make had a goat.

There was considerable debate among fans of the show (as well as statisticians and mathematicians) about the following question: When Monty shows you that there's a goat behind one of the doors that you didn't pick, should you switch your choice or should you stay with your original decision?

2 Programming Assignment

You will write a program `monty.cpp` to simulate this game and analyze this situation.

Develop a program `monty.cpp` in C++. It takes one command line argument. This argument is the number of times to run the experiment. Print out percentages of how many switches win and how many stay win.

Generate a random set-up of the car and goats. Have a player pick a door at random. Look behind the scenes to see if the player should switch doors or stay. Report how many times they should **stay** and how many times they should **switch**.

After running a few tests, decide what is the best strategy: Switch or Stay? At the top of your `monty.cpp` file, tell us which strategy you picked and explain why in a comment.

No threads are required to solve this problem. It is an **introduction to C++** question, not a concurrency question.

Optional Bonus 10 Points: use two threads each running half the tests. Combine the two totals together. (Note: since running a test is reasonably fast, you might not see any speed improvement. Your code might actually take longer this way!)

2.1 Examples

Note: Due to randomization, don't expect to get these numbers exactly. As the number of tests increases, you should approach a consistent answer. None of the example tests below are large enough to see the pattern.

```
% g++ -o bin/monty src/monty.cpp
% ./bin/monty
Monty Hall Problem Simulator
Usage: monty num_tests
% ./bin/monty cat
Monty Hall Problem Simulator
Number of Tests is not a number.
% ./bin/monty 2
Monty Hall Problem Simulator
Switch would win 50.00 percent of experiments.
Stay would win 50.00 percent of experiments.
% ./bin/monty 5
Monty Hall Problem Simulator
Switch would win 80.00 percent of experiments.
Stay would win 20.00 percent of experiments.
% ./bin/monty 16
Monty Hall Problem Simulator
Switch would win 75.00 percent of experiments.
Stay would win 25.00 percent of experiments.
% ./bin/monty 21
Monty Hall Problem Simulator
Switch would win 71.43 percent of experiments.
```

2.2 Implementation

You are expected to write professional code. Use good variable and function names. Provide comments explaining how the code works. Documents each function in the header file with comments. You will be graded on style as well as functionality.

2.3 Citations

If you use any outside resources, talk about algorithm design with other students, or get help on assignments you **must** cite your resources in the comments. Uncited sources are an Academic Honesty Violation. Cited sources may lead to a deduction depending on the amount of code used, but will not violate Academic Honesty Policies.

You are expect to write the majority of the code yourself and use resources for things like looking up commands. For example, if you forgot how to test if a file can be opened for reading you could look it up and cite a source. If you copy a critical algorithm and cite the code, you may still get a deduction for not developing the code yourself.

2.4 Makefile

You **must** provide a makefile to compile your code. We will type make monty and it **must** build your program. If there are any compile errors or a makefile is not provided we cannot test your code.

These images are shown in the lecture slides.

You must have the following make targets:

1. make monty - Builds the Program
2. make none - ./bin/monty
3. make cat - ./bin/monty cat
4. make test16 - ./bin/monty 16
5. make test100 - ../bin/monty 100
6. make test2000 - ../bin/monty 2000
7. make test10000 - ./bin/monty 10000
8. make clean - Remove compiled code.

2.5 Other Requirements

If you submission does not meet the following guidelines we will not be able to grade it.

- You **must** use the C++ 17 Standard threads. No other thread libraries (pthreads, boost, etc) may be used. `https://en.cppreference.com/w/cpp/header/thread`
- Code **must** run on tux and be compiled with g++.
- All code **must** compile using the C++ 17 or above standard. (−−std=c++17)
- All code **must** be submitted in a single zip file. (No tar.gz, rar, etc)
- A working makefile **must** be provided.
- You may use libraries in the C++ 17 standard unless noted elsewhere in the instructions. `https://en.cppreference.com/w/cpp/header`
- Your code **must** compile. You should always submit stable code, we will not debug code that does not compile.

3 Grading

This homework is worth 50 points.

Since this program uses randomization, two runs will almost always be different. Your output should be similar to the expected results over multiple tests, even though any one tests may be different. The larger the input the more consistent the tests will become. For example, ./bin/monty 2 should be very inconsistent but ./bin/monty 2000 should be very consistent between runs.

This randomization is why the examples above are with small numbers, but the requested tests use large numbers. We don't want to give away the pattern. You have to figure it out.

Question 1 : 2 points
Name, Date in comments at top of file (use doxygen @file)

Question 2 : 3 points
Code uses well designed functions (including doxygen function comments on all function)

Question 3 : 5 points
Answer to written question in comments

Question 4 : 10 points
Overall Code Design

Question 5 : 5 points
Output Matches Expectations for make none

Question 6 : 5 points
Output Matches Expectations for make cat

Question 7 : 5 points
Output Matches Expectations for make test16

Question 8 : 5 points
Output Matches Expectations for make test100

Question 9 : 5 points
Output Matches Expectations for make test2000

Question 10 : 5 points
Output Matches Expectations for make test10000

Question 11 : 0 points
Extra Credit: 10 point bonus